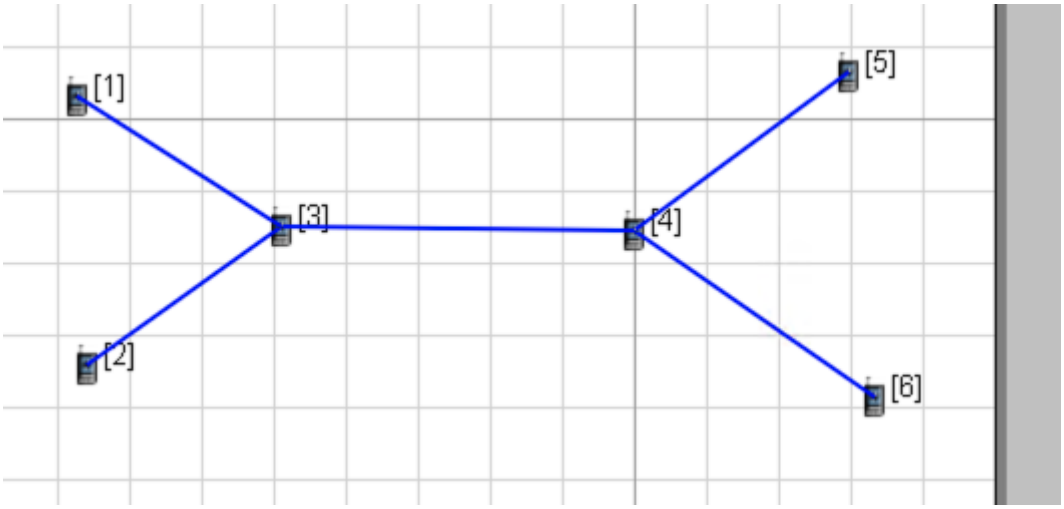
EXata 扩展（四）：HTTP

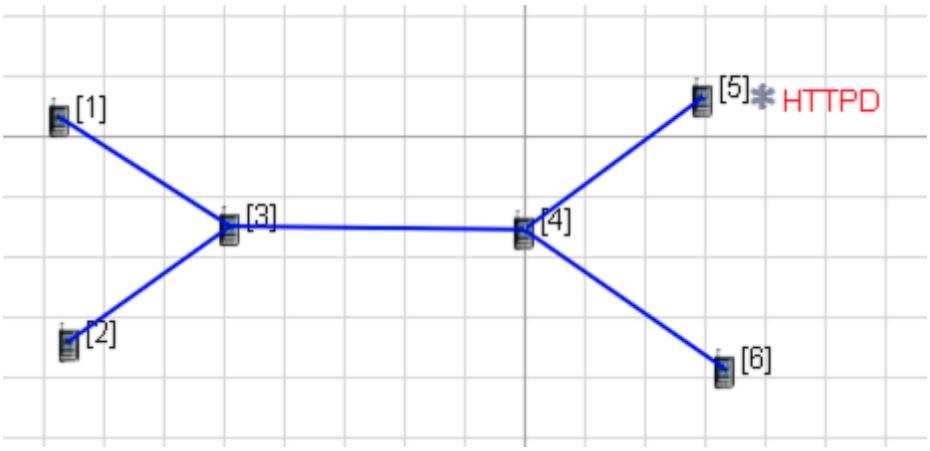
目标：了解 EXata 中 HTTP 的实现方式，为实现 in-network cache 做准备

1. HTTP 场景

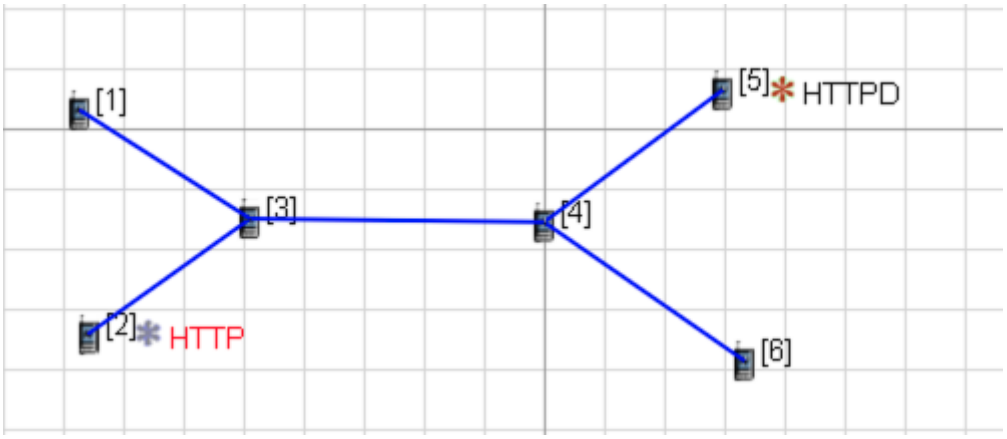
1. 创建一个有线瓶颈场景，命名为ex_http.config;



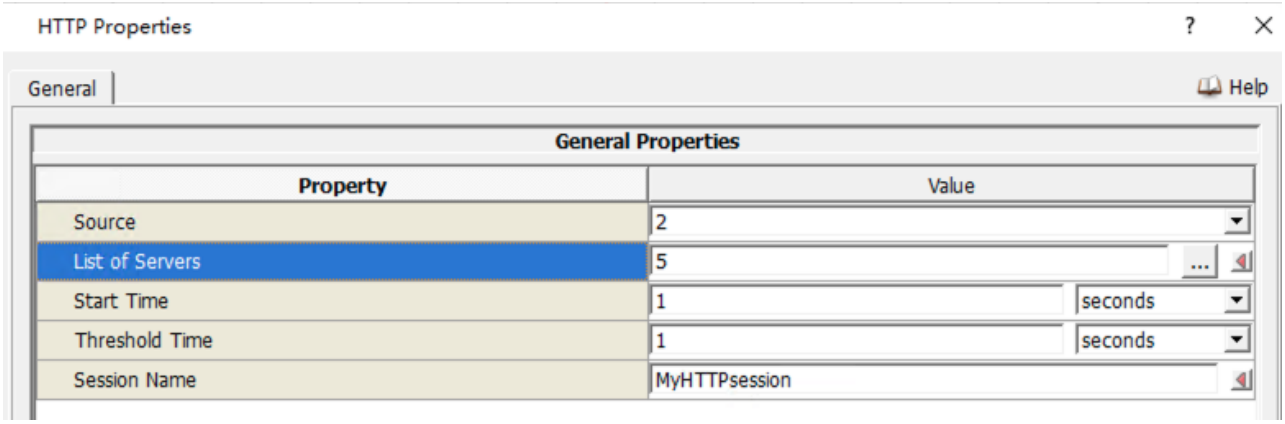
2. **安装 HTTP 服务器**：点击Single Host Applications 中的 HTTPD，安装在节点5



3. 安装 HTTP 客户端：点击Single Host Applications 中的 HTTP，安装在节点2；



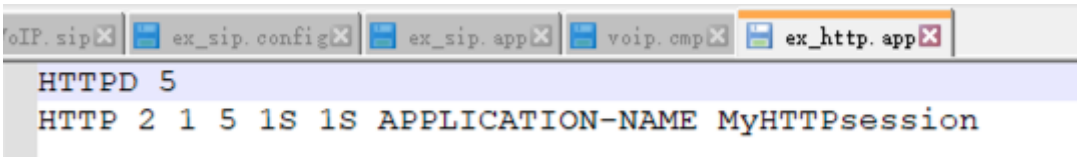
4. **客户端属性配置**：打开 HTTP 属性框，在“List of Servers”中，输入 HTTPD 的节点 ID “5”或其 IP 地址，设置“Threshold Time”即 （think time）；在“Session Name ”输入“My HTTP session”。



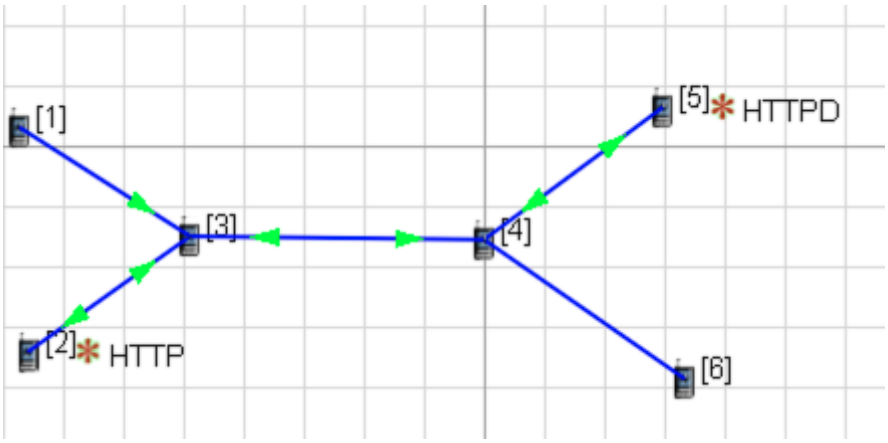
5. 查看应用配置文件：ex_http.app。注意：HTTP 客户端中第2个参数“1”表示HTTP Servers 的个数。

To configure an HTTP client, use the following format:

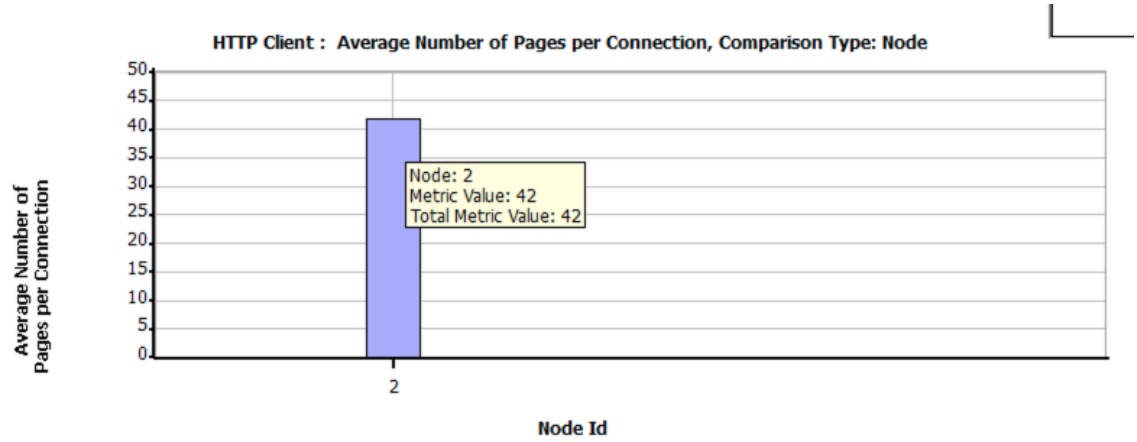
```
HTTP  <client-ID> <num-servers> <server-1> ... <server-n>
      <start-time> <thresh>
      [APPLICATION-NAME <application-name>]
```

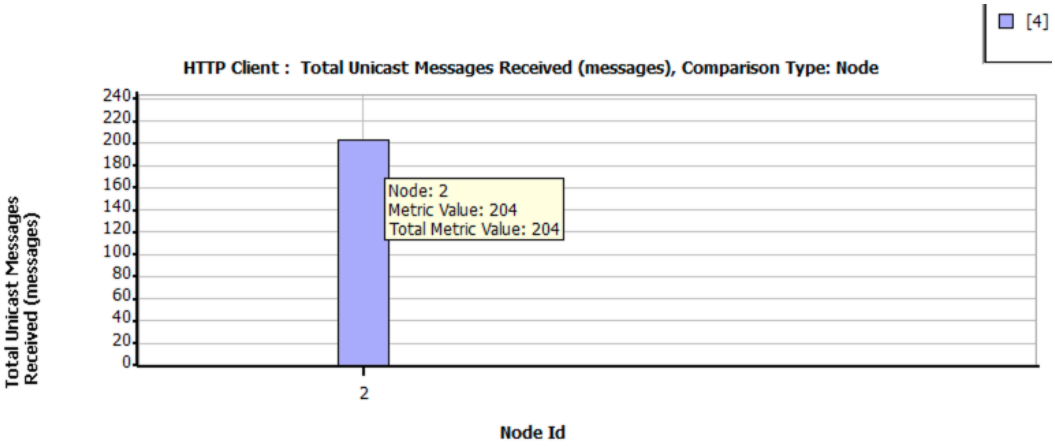


6. 运行场景

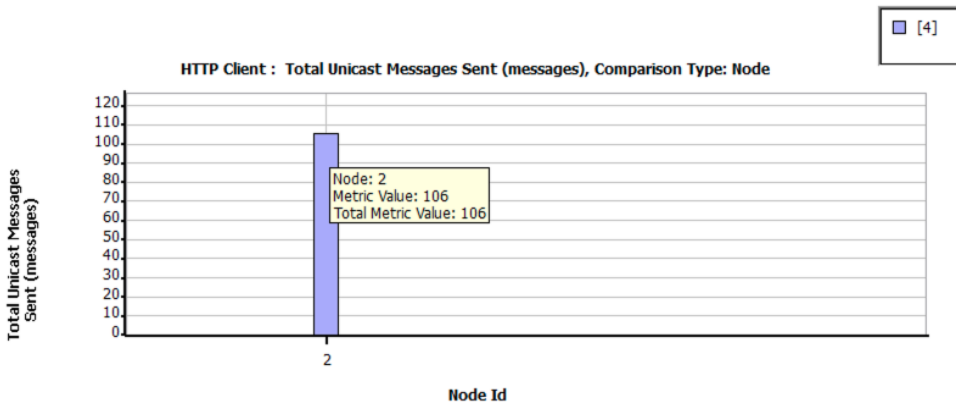


7. 客户端（Node：2）结果分析

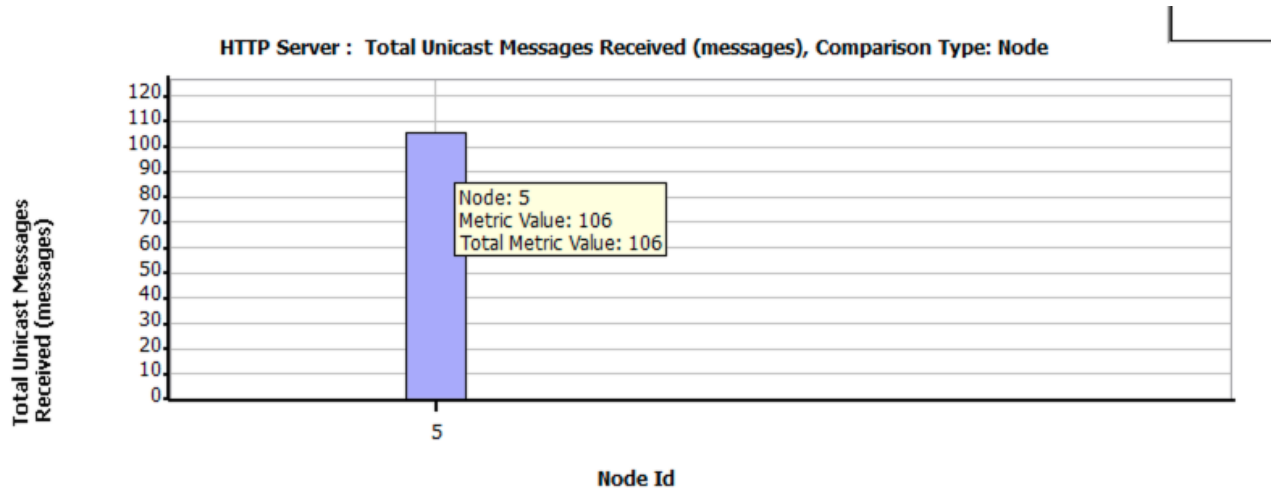


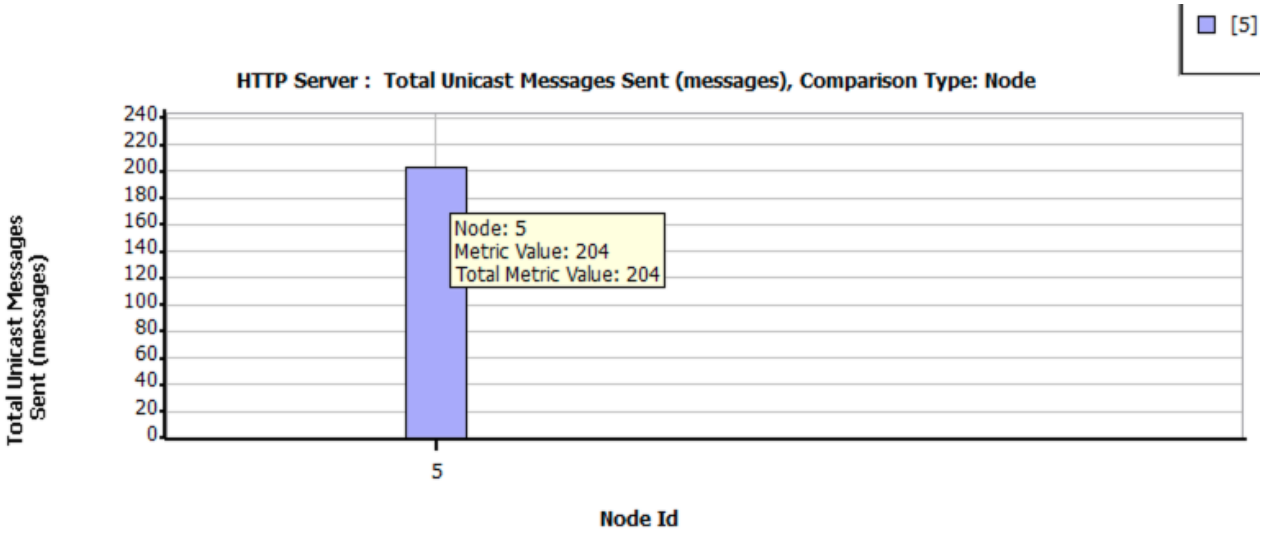


8.



9. 服务器（Node: 5）端结果分析





2. EXata 中 HTTP 的实现

在 ex_http.app中有两个应用：HTTPD 和 HTTP，前者对应 Server，后者对应 Client。对应在 *APP_InitializeApplications* 中对应两个应用的初始化：HTTP 和 HTTPD

2.1 HTTPD 初始化

即 appStr == “HTTPD” 时，

1. 利用 *sscanf* 读取 HTTPD 的参数，即服务节点 id，（i.e. 5）；
2. 取服务节点指针：*MAPPING_GetNodePtrFromHash*,
3. HTTP Server初始化：*AppHttpServerInit*。这里完成以下工作：
    a. 取网络接口类型：IPV4_ONLY? ： *MAPPING_GetNetworkProtocolTypeForInterface*;
    b. 取接口地址：MAPPING_GetInterfaceAddrForNodeIdAndIntfId;
    c. 取节点的 IP Data：NetworkDataIp* ip = node->networkData.networkVar;
    d. 添加 HTTP Server 实例：先用 httpServer = AppHttpServerGetServerForThisInterfaceIndex(node, i); 提取；如空，则注册新实例info： httpServer = AppHttpServerGetServerForThisInterfaceIndex(node, i);
    e. 回调函数地址变化处理函数更新【添加 HTTP 事件处理函数指针，当地址变化时调用该函数】：*NetworkIpAddAddressChangedHandlerFunction* 【network_ip.cpp】；
    f. 【重要!!】打开 TCP 连接，监听默认端口(默认：8080)：node->appData.appTrafficSender->appTcpListen 【app_trafficSender.cpp】；

2.2 HTTP 初始化

即 appStr == “HTTP” 时，

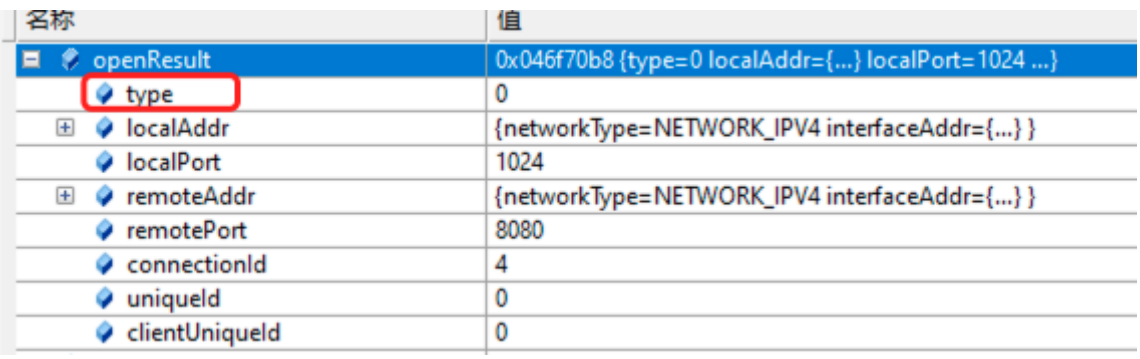
1. 利用 *sscanf* 读取 HTTP 的参数，如：HTTP 2 1 5 1S 1S;
2. 取客户端节点指针：*MAPPING_GetNodePtrFromHash;*
3. *读取服务器个数（i.e. 1）：* token = IO_GetDelimitedToken(iotoken, next, delims, &next);
4. 读取服务器 id： IO_AppParseDestString

5. 检测是否有 url：false；【http 客户端配置时可以指定服务器节点 id，或输入 url】【但 HTTP 消息应该必须有 URL 的，需要 Wlreshark 抓包确认 EXata 有没有实现；下次，也可以加上 DNS 一起实现，就会更加完整】
6. 读取并设定下一个参数–启动时间：startTime （i.e.1S）：
7. 读取并设定下一个参数–思考时间或threshold：threshTime；
8. 释放临时动态分配缓存；
9. HTTP Client初始化：*AppHttpClientInit*。这里要完成以下工作：
    a. 创建新的 HTTP Client：*AppHttpClientNewHttpClient*；
        i. 注册新应用，添加到节点的应用列表中：*APP_RegisterNewApp*(node, APP_HTTP_CLIENT, httpClient);【app_util.cpp】
    b. 设定客户端参数：threshold、servers、num_servers、Zipf_constant（记得：目前 http是个流量发生器）。
    c. 设定各发往各服务器请求的 Zipf 常数：Zipf_constant；【注意：在哪里舍得时间间隔？】
    d. 初始化统计量： clientPtr->stats.itemRequestBytes = 0;
    clientPtr->stats.pageItems = 0;
    e. 设定远端地址：单服务器、dynamic address
    f. 创建新的APP 统计量，并进行初始化： *clientPtr->appStats = new STAT_AppStatistics*； *clientPtr->appStats->Initialize*;
    g. 更新 Client 地址信息：*AppHttpAddAddressInformation(node, clientPtr)*;
    h. 重要!! 新建客户端 TCP连接：node->appData.appTrafficSender->*appTcpOpenConnection*;

2.3 HTTP 客户端的事件处理

HTTP 客户端的事件处理同样遵循EXata 的事件处理层次：节点-->层-->协议

1. 节点：NODE_ProcessEvent
2. 应用层：APP_ProcessEvent，读取协议类型 protocolType，调用 *App_GetProtocolType*;
    a. 如果是APP_HTTP_CLIENT 协议（#81），则触发 AppLayerHttpClient；
    b. 如果是 APP_HTTP_SERVER 协议（#8080），则触发 AppLayerHttpServer;【这是模拟 HTTP Server 消息处理过程的核心】
3. ***AppLayerHttpClient***，【这是模拟 HTTP Client 消息处理过程的核心】根据消息类型处理
    a. *MSG_APP_FromTransOpenResult （#601）：从传输层上报到 Open Connection 的结果*
        i. *把 msg 转化成* TransportToAppOpenResult 类型多 openResult;



ii. 检查 TCP 连接是否正常打开：assert(openResult->type == *TCP_CONN_ACTIVE_OPEN*);

CP_CONN_ACTIVE_OPEN 即 type := 0;

iii. *Client 更新状态：* clientPtr = ***AppHttpClientUpdateHttpClient***(node, openResult);

iv. 返回当前页面的 item 数：clientPtr->stats.pageItems = ***AppHttpClientDetermineItemCount***(clientPtr);【默认，EXata 都从实际流量中总结到的随机分布分配】

v. 计算 primary request 的字节数：AppHttpClientDeterminePrimaryRequestLength；【默认，EXata 都从实际流量中总结到的随机分布分配】

vi. 发送主请求 Primary Request 到 Server：***AppHttpClientSendPrimaryRequest***;

1. 在净荷最后一个字节赋值为 'p'：*(payload + sendSize – 1) = 'p';
2. **HTTP client 状态**设为"WAIT_PRIMARY_RESPONSE"：clientPtr->state = WAIT_PRIMARY_RESPONSE; HTTP 客户端一共有 5 个 状态：

```
typedef
enum
{
    IDLE,
    XMIT_PRIMARY_REQUEST,
    XMIT_SECONDARY_REQUEST,
    WAIT_PRIMARY_RESPONSE,
    WAIT_SECONDARY_RESPONSE
}
HttpClientState;
```

3. **设定SendWaitReplyTimer**：AppHttpClientSendWaitReplyTimer(node, clientPtr, WAIT_PRIMARY_REPLY_TIMER);【这是下一个要处理的消息】
4. 创建 TCP 消息
5. 装载 TCP 净荷
6. TCP 发送